

Coloring so that no Pythagorean Triple is Monochromatic

Joshua Cooper and Ralph Overstreet

University of South Carolina, Department of Mathematics,
1523 Greene Street, Columbia, SC 29208
cooper@math.sc.edu, overstrrr@email.sc.edu
<http://www.math.sc.edu/>

Abstract. We address the question of the “partition regularity” of the Pythagorean equation $a^2+b^2 = c^2$; in particular, can the natural numbers be assigned a 2-coloring, so that no Pythagorean triple (i.e., a solution to the equation) is monochromatic? We prove that the hypergraph of Pythagorean triples can contain no Steiner triple systems, a natural obstruction to 2-colorability. Then, after transforming the question into one about 3-CNF satisfiability and applying some reductions, a SAT solver is used to find a 2-coloring for $\{1, \dots, 7664\}$. Work continues as we seek to improve the reductions and extend the computation.

Keywords: CNF, Linear 3-Uniform Hypergraph, Partial Triple System, Partition Regular, Pythagorean Triple, Ramsey Theory, SAT, Steiner Triple System

1 Introduction

One of the central problems in Ramsey Theory on the integers (see, for example, [1]) is the following question: given an equation – or set of equations – and a positive integer k , does there exist a coloring of \mathbb{N} so that there are no monochromatic solutions to the equation(s)? That is, suppose the family \mathcal{F} of subsets of \mathbb{N} is defined to be the set of solutions $\{x_1, \dots, x_n\}$ to a system of equations in n variables. Does there exist a function $c : \mathbb{N} \rightarrow [k]$ so that, for each $F \in \mathcal{F}$, there are $x_i, x_j \in F$ with $c(x_i) \neq c(x_j)$? A system of equations is called “partition regular” if, for any number of colors k , every k -coloring of the integers admits a monochromatic solution. Perhaps the most famous examples of partition regularity are the equations $x + y = z$ (Schur’s Theorem) and $x + y = 2z$ (van der Waerden’s Theorem); Rado’s Theorem provides a vast generalization of these examples.

A question that is surprisingly resistant to extant methods is the partition regularity of the “Pythagorean equation” $x^2 + y^2 = z^2$; see, e.g., [2]. To date, it is not even known if it is possible to 2-color the naturals so that there are no monochromatic Pythagorean triples. However, there is a natural translation of

this question into that of solving a certain 3-CNF satisfiability problem:

$$\bigwedge_{\substack{i,j,k \\ i^2+j^2=k^2}} (x_i \vee x_j \vee x_k) \wedge (\neg x_i \vee \neg x_j \vee \neg x_k). \quad (1)$$

In this case, “true” and “false” assignments are the colors given to the indices of variables.

In order to show that the Pythagorean triples are partition regular, it suffices to demonstrate a subfamily of them which is not bipartite, i.e., any 2-coloring induces a monochromatic triple. (The existence of such a family follows from the de Bruijn-Erdős Theorem, q.v. [3].) It is known that no two Pythagorean triples share two points, i.e., no leg and hypotenuse of one right triangle are the two legs of another integer right triangle. Indeed, the two equations $x^2 + y^2 = z^2$ and $y^2 + z^2 = t^2$ yield two squares (y^2 and z^2) whose sum and difference are also squares. One can find an elementary proof that no such pair of squares exists in Sierpiński’s classic text [4].

Therefore, in our search for nonbipartite subsystems of triples, we need only consider those in which each pair of triples intersect in at most one point. Such families are known as “linear 3-uniform hypergraphs” (in the graph theory community) and “partial triple systems” or “packings” (in design theory). The smallest nonbipartite partial triple system is the ubiquitous Fano plane F_7 , the projective plane over \mathbb{F}_2 with seven points and seven triples. F_7 is also a “Steiner triple system,” meaning that each pair of points is contained in exactly one triple. In fact, it is no hard to see that every Steiner triple system is not bipartite, so it is reasonable to search for these classic obstructions to 2-colorability among the Pythagorean triples. However, we show below that a broad class of triple systems including the family PYTH of all Pythagorean triples contains no Steiner triple systems. Therefore, any search for nonbipartite partial triple systems in PYTH necessarily must consider other systems than Steiner triple systems.

In the next section, we introduce some notation and definitions. Section 3 contains the proof that PYTH, and all ordered triple systems with the “sum property”, contain no Steiner triple systems. In the following sections, we describe the satisfiability solving methodology used to show that at least [7664] is 2-colorable without any monochromatic Pythagorean triples. We conclude with some directions for further research.

2 Preliminaries

A *triple system* \mathcal{H} is a pair $(V, E) = (V(\mathcal{H}), E(\mathcal{H}))$ consisting of a vertex set V and a family of unordered triples $E \subset \binom{V}{3}$. A *partial triple system*, aka a *3-uniform linear hypergraph*, is a triple system in which each pair of distinct edges intersect in at most one vertex. A *Steiner triple system* is a partial triple system \mathcal{S} so that, for each pair of vertices x and y , there is some z so that $\{x, y, z\}$ is an edge of \mathcal{S} . An *ordered triple system* \mathcal{H} is a triple $(V, E, <) = (V(\mathcal{H}), E(\mathcal{H}), <_{\mathcal{H}})$ consisting of a vertex set V , a family of unordered triples $E \subset \binom{V}{3}$, and a total

ordering $<$ on V . We say that an ordered triple system \mathcal{H} has the *sum property* if, whenever $\{a, b, c\}$ and $\{a', b', c'\}$ are two edges with respective maxima c and c' ,

$$(a \leq a') \wedge (b < b') \Rightarrow (c < c'). \quad (2)$$

Examples:

1. Let $V = \mathbb{Z}$, and, for each $a \neq b$, let $e = (a, b, a + b)$ be an edge of E . We call this the *Schur Triple System*, and denote it by SCHUR. It is easy to see that SCHUR has the sum property. This shows immediately that the sum property does not imply finite colorability, since Schur's Theorem says that any coloring $\chi : \mathbb{N} \rightarrow [c]$ with finitely many colors c admits a monochromatic $e \in \text{SCHUR}$, i.e., $|\chi(e)| = 1$. Such a map is called a (weak) hypergraph coloring. A map $\chi : \mathbb{N} \rightarrow [c]$ so that $|\chi(e)| = 3$ instead of just $|\chi(e)| > 1$ is known as a “strong coloring,” and corresponds exactly to a proper vertex coloring of the complement of the “leave,” i.e., the graph of all pairs contained in some triple $e \in E(\mathcal{H})$. Some areas of mathematics call this the “shadow” graph or “1-skeleton” of \mathcal{H} , and denote it by the boundary operator $\partial\mathcal{H}$.
2. It is clear that any order-preserving isomorphic image of a triple system with the sum property also has the sum property, and that any subhypergraph of a triple system with the sum property does as well. For example, we define the *Pythagorean Triple System* PYTH by $V(\text{PYTH}) = \mathbb{N}$ and $E(\text{PYTH}) = \{\{a, b, c\} : a^2 + b^2 = c^2\}$. Since one can embed PYTH into SCHUR monotonically by the map $n \mapsto n^2$, PYTH has the sum property as well. It is a wide open problem to determine whether PYTH has a weak coloring with finitely many colors. (It is even open whether it is strongly colorable.) As mentioned in the introduction, PYTH is actually linear, i.e., no two edges intersect in more than one vertex.
3. A special subsystem of PYTH is the *Primitive Pythagorean Triple System* PRIM consisting of all Pythagorean triples which are relatively prime. That is, $V = \mathbb{N}$ and

$$E(\text{PRIM}) = \{\{a, b, c\} : a^2 + b^2 = c^2 \text{ and } \gcd(a, b, c) = 1\}. \quad (3)$$

It is easy to see that PYTH is actually a union of dilates of PRIM by each $d \in \mathbb{N}$. However, PRIM is bipartite: the parity coloring $n \mapsto n \pmod{2}$ provides a 2-coloring.

We define a special class of partial triple systems called “bicycles.” The k -bicycle has $2k + 2$ vertices and $2k$ edges. Its vertices are the elements of \mathbb{Z}_{2k} and two “antipodes” a and b ; its edges are all triples of the form $\{a, 2j, 2j + 1\}$ and $\{b, 2j - 1, 2j\}$, $0 \leq j < k$. The 2-bicycle is also known as the *Pasch configuration*, or *quadrilateral*: the six-point partial triple system consisting of the edges abc , ade , bef , cdf . The 3-bicycle appears in the literature as the “hexagon” (e.g., [5]): eight points $\{a, b, d, e, f, g, h, i\}$ with edges $\{afh, aei, adg, beh, bdi, bfg\}$.

The following proposition follows immediately from well-known results in the theory of triple systems. (See, for example, [6].) For completeness, we give a short proof.

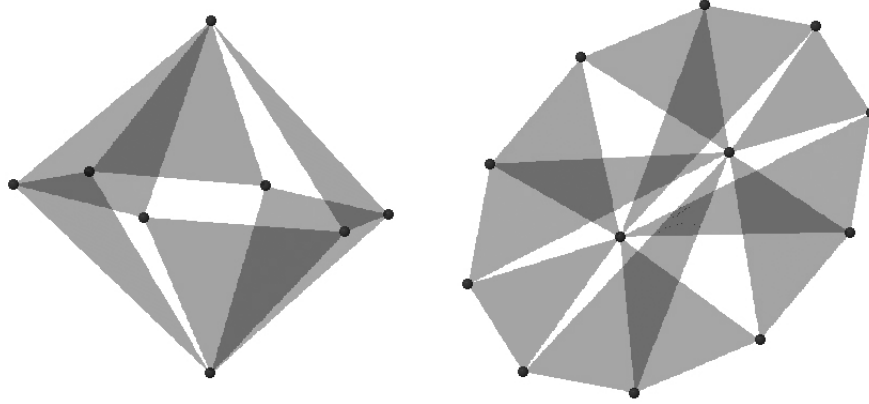


Fig. 1. A 3-bicycle and a 5-bicycle.

Lemma 1. *If v, w are vertices of nontrivial Steiner triple system \mathcal{S} , then there is a k -bicycle for some $k \geq 2$ in \mathcal{S} with antipodes v and w .*

Proof. Define the “link” \mathcal{S}_x of a vertex $x \in \mathcal{S}$ to be the set of pairs $\{a, b\}$ so that $\{a, b, x\}$ is an edge of \mathcal{S} . For some $z \in \mathcal{S}$, $\{v, w, z\}$ is a triple of \mathcal{S} , so \mathcal{S}_v consists of a perfect matching M_1 on $\mathcal{S} - \{v, w, z\}$, plus the edge $\{w, z\}$; similarly, \mathcal{S}_w consists of a perfect matching M_2 on $\mathcal{S} - \{v, w, z\}$, plus the edge $\{v, z\}$. The union of M_1 and M_2 is composed of even-length cycles of length at least 4; any one of these forms a bicycle with antipodes v and w .

We now define two weaker implicants of the sum property. We say that an ordered triple system has the *upper sum property* if, whenever $\{a, b, c\}$ and $\{a', b', c'\}$ are two edges with c and c' their respective maxima,

$$(b > b') \Rightarrow (c > c'). \quad (4)$$

This clearly follows from the sum property by setting $a = a'$. We say that an ordered triple system has the *lower sum property* if, whenever $\{a, b, c\}$ and $\{a', b', c\}$ are two edges with c as both of their maxima,

$$(a > a') \Rightarrow (b < b'). \quad (5)$$

To see that the lower sum property follows from the sum property, suppose that $a > a'$ but $b > b'$. Then it follows that the maximal element of $\{a, b, c\}$ is greater than the maximal element of $\{a', b', c\}$, whence $c > c$, a contradiction. We say that two pairs of integers $\{a, b\}$ and $\{c, d\}$ are “nesting” if $a < c < d < b$. Then it is possible to restate the upper sum property as the fact that, for each vertex x , the subset of the link graph \mathcal{H}_x intersecting $\{y : y \geq x\}$ is a non-nesting matching. The lower sum property may be similarly restated as the fact that,

for each vertex x , the subset of the link graph \mathcal{H}_x induced by $\{y : y \leq x\}$ is a fully nested matching, i.e., for each two edges e and f , e is nested in f or vice versa.

3 Sum Property implies No STS

Proposition 1 *If \mathcal{H} has the upper sum property, and \mathcal{Q} is a k -bicycle in \mathcal{H} , then the maximal two points of \mathcal{Q} are not its antipodes.*

Proof. Let

$$\mathcal{Q} = (\mathbb{Z}_{2k} \cup \{a, b\}, \{a01, a23, a45, \dots\} \cup \{b12, b34, b56, \dots\}) \subset \mathcal{H} \quad (6)$$

be a k -bicycle, and suppose a and b are the maximal two points of \mathcal{Q} . We may assume without loss of generality that $a > b$. Then the maximal elements of all triples are a or b (depending on which antipode they contain). Therefore, since $\{a, 2j, 2j+1\} \cap \{b, 2j+1, 2j+2\} = \{2j+1\}$,

$$(a > b) \Rightarrow (2j > 2j+2) \quad (7)$$

for each $0 \leq j < k$. However, the quantities above are modulo $2k$, whence the set of resulting inequalities is circular and therefore inconsistent.

Proposition 2 *If \mathcal{H} has the lower sum property, and \mathcal{Q} is a k -bicycle in \mathcal{H} , then the maximal two points of \mathcal{Q} are not its antipodes.*

Proof. Let

$$\mathcal{Q} = (\mathbb{Z}_{2k} \cup \{a, b\}, \{a01, a23, a45, \dots\} \cup \{b12, b34, b56, \dots\}) \subset \mathcal{H} \quad (8)$$

be a k -bicycle, and suppose a and b are the maximal two points of \mathcal{Q} . We may assume without loss of generality that $a > b$. Then the maximal elements of all triples are a and b (depending on which antipode they contain). Since $\{a, 2j, 2j+1\}$ is an edge for each $0 \leq j < k$, the k pairs $\{2j, 2j+1\}$ are a matching, and they are linear ordered by nesting, by the lower sum property. Suppose, without loss of generality, that $\{0, 1\}$ is the outermost matching edge and $0 < 1$, so that, for all $x \in \mathbb{Z}_{2k} \setminus \{0, 1\}$, $0 < x < 1$. The pairs $\{-1, 0\}$ and $\{1, 2\}$ are also nested, since they arise from the edges $\{b, -1, 0\}$ and $\{b, 1, 2\}$. Hence, $-1 < \{1, 2\} < 0$ or $1 < \{-1, 0\} < 2$, and each of these possibilities contradicts $0 < 1$.

Corollary 3 *If \mathcal{H} has the full, lower, or upper sum property, then it does not contain any Steiner triple system. In particular, PYTH contains no Steiner triple system.*

Proof. Suppose \mathcal{H} contained some Steiner triple system \mathcal{T} . Let a and b be the maximal elements of \mathcal{T} . Then a and b are the antipodes of some bicycle, by Lemma 1. However, this contradicts Propositions 1 and/or 2.

Note that a triple system with the sum property *can* contain a quadrilateral: for example, SCHUR contains $\{5, 15, 20\}$, $\{5, 8, 13\}$, $\{7, 8, 15\}$, $\{7, 13, 20\}$.

4 The Computational Approach

We now describe the methodology used to certify the 2-colorability of the induced hypergraph of PYTH on the vertex set [7664].

4.1 Listing the Triples up to N

To make a list of all Pythagorean triples having only elements less than or equal to some upper bound N , we use a function based on Dickson's Method, from Wikipedia [7]. Our function for generating triples uses the equations $r^2 = 2st$ for r, s , and $t \in \mathbb{N}$, and $x = r + s, y = r + t$ and $z = r + s + t$, with $x^2 + y^2 = z^2$. With these equations, we step through all combinations of s and t so that $r \in \mathbb{N}$ and $z \leq N$. The Python code is short.

```
import numpy as np
def dicksonGenTriples(uBound):
    data = []
    sSteps = range(1,uBound)
    for s in sSteps:
        tSteps = range(s,uBound-s)
        for t in tSteps:
            r = np.sqrt(2*s*t)
            if r == np.round(r):
                r = int(r)
                z = r + s + t
                if z >= uBound:
                    break
                x = r + s
                y = r + t
                if [x,y,z] not in data:
                    data.append([x,y,z])
    with open('triples-dickson.json','w') as f:
        json.dump(data, f)
    return data
```

4.2 Writing the CNF

We remapped the integers involved in the Pythagorean triples to consecutive integers starting with 1 since, some SAT solvers require it. The bijection for remapping is represented in the Python code by a dictionary.

```
{"3": 1, "4": 2, "5": 3, "6": 4, "8": 5, "10": 6}
```

The following is a CNF file for the triples with integers up to 10. These are [3,4,5] and [6,8,10].

```

c 10
p cnf 6 4
1 2 3 0
-1 -2 -3 0
4 5 6 0
-4 -5 -6 0

```

A full specification for CNF (Conjunctive Normal Form) is given at [8].

4.3 Choosing a Solver

We experimented with glueSplit.clasp, Dimetheus, SWDiA5BY, PeneLoPe, Lingeling, and Treengeling, all from <http://www.satcompetition.org/> in 2014. Treengeling is a parallel solver that runs on some or all of the cores of a single machine, and seemed to be the fastest for a CNF corresponding to the set of triples up to about 7000. These runs completed within a few hours on each of the many solvers. We did not emphasize finding the best solver.

4.4 Reducing the Size of the Input to the Solver

A natural goal of SAT solving is to ease the burden on the solver by finding parts of the hypergraph, corresponding to the solver’s original CNF, which can be safely removed without changing the satisfiability of the new, reduced CNF. We have had to rely on these reductions for upper bounds above 7620.

Our only success to date at reducing the size of the input to the solver in this way, is “pendant removal”. By pendant, we mean any edge (triple) of the 3-uniform hypergraph which has at least one vertex not contained in any other edges. Once a triple system with a pendant removed is properly 2-colored, the pendant’s vertex, or vertices, of degree one can then be assigned any color that is not assigned to some other vertex of its edge (of which there is always at least one). Pendants are always removable and can be removed iteratively until no more pendants remain. Once these pendants have been removed from a graph and the remaining graph has been fed to the solver and colored, it is easy to add back the removed pendant edges and 2-color them as they are added back. The edges are added back in the reverse order that they were removed.

5 Parallelizing Across Multiple Computer Cluster Nodes

5.1 The Overall Strategy

We use a strategy to parallelize across multiple nodes of a cluster similar to one used in a paper by Biere, Heule, Kullmann, and Wieringa [9]. In particular, we convert the problem into 2^m many potentially easier problems, by making 2^m copies of the original CNF file and appending a distinct list of extra clauses to each one. We run these new CNF files on up to 2^m cluster nodes; one node per CNF. As soon as any node completes, if that CNF is satisfiable, we have our

solution and can force all the other nodes to terminate before completing. The various nodes do not communicate amongst themselves and run independently from one another.

To create the extra clauses for each of the 2^m copies of the original CNF file, we select m special vertices (integers), each coming from some Pythagorean triple which was used to generate the original CNF. For each special vertex, i , we make a new clause

$$(\neg x_i \vee \neg x_i \vee \neg x_i) \quad (9)$$

or

$$(x_i \vee x_i \vee x_i). \quad (10)$$

This gives us 2^m different lists of m extra clauses to add to each of the 2^m copies of the original CNF file. These extra m clauses provide the solver with predetermined truth assignments for specific vertices. Either one of these 2^m runs of the SAT solver, on the slightly varying CNF files, will complete and provide a valid coloring, or all the runs will complete and each certify unsatisfiability – given sufficient computational time.

5.2 An Example CNF File Modified for Parallelization

We present one of the 4 CNF files for the list of triples $[[3,4,5],[6,8,10]]$, split on 2 vertices; 3 and 4, which are remapped to 1 and 2. The line which begins with p is also updated by adding $m = 2$ to the last integer in the original CNF file on that line. The original CNF had a 4 in that spot. Note especially the last 2 clauses.

```
c 10
p cnf 6 6
1 2 3 0
-1 -2 -3 0
4 5 6 0
-4 -5 -6 0
1 1 1 0
-2 -2 -2 0
```

5.3 Choosing Vertices to Split On

It would be desirable to choose a list of m special vertices so that each vertex in the list is relatively independent from the others. By independence, we mean that any assignment of truth values to the variables corresponding to special vertices leads to a similar number of (partial) satisfying assignments. We have attempted to gauge independence by measuring the time needed for each of the 2^m truth value combinations to run using Treengeling. The minimum distance in the hypergraph of all Pythagorean triples, up to an upper bound, N , between any two of the special vertices is a reasonable proxy for their independence. Under current technical limitations, we have found $m = 2$ to run the fastest.

The BFS Method of Finding Distant Vertices A promising method of choosing some v_1 and v_2 which are distant from one another in the hypergraph of Pythagorean triples is to use what we call the breadth-first search (BFS) method. By way of illustration, we present Figure 2 and Table 1. We start from a seed triple t_0 , and use the method to find a t_{11} that is furthest from t_0 . Once a

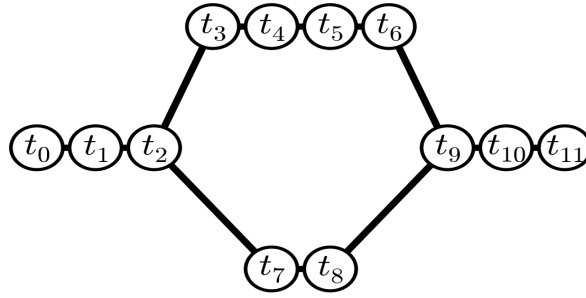


Fig. 2. A graph of some hypothetical list of triples where any 2 triples share an edge if they intersect. For example $t_0 = [3,4,5]$ and $t_1 = [5,12,13]$ share an edge in a graph of this type, as the 2 triples share a vertex.

Table 1. The triples of each BFS level corresponding to the graph in Figure 2.

level	0	1	2	3	4	5	6	7
	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_{11}
				t_7	t_8	t_9	t_{10}	

triple appears in a level, it will not appear again in the list. When there is more than one path from the seed to some other triple, the listing in Table 1 shows only the shortest path. Thus any two triples which are in two distant levels from one another, necessarily have a high minimum graph distance between them.

In the Pythagorean Triple System, to find two vertices that are relatively far apart, we choose a seed triple with minimum constituent integers as small as possible. In the example of Figure 3, we run the BFS method on a list of triples that has had its pendants removed. The plot gives an indication of the proximity of two vertices to each other and for the connectedness, loosely defined, of each of the two vertices to the rest of the hypergraph.

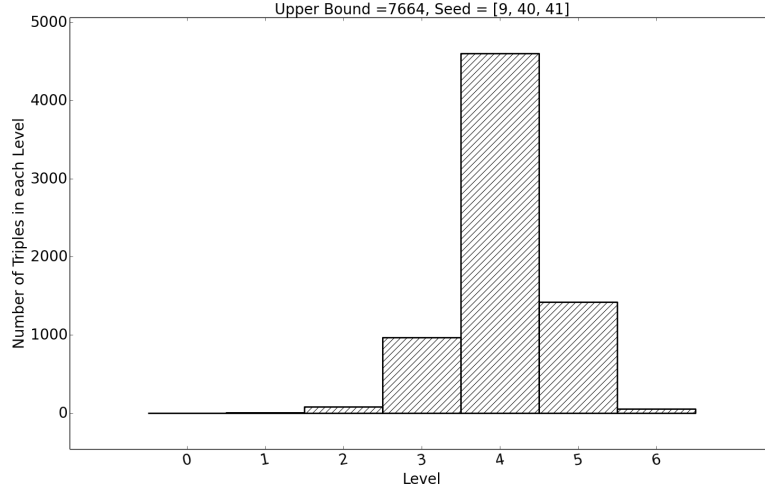


Fig. 3. The bar plot shows the size of each BFS level for the list of pendant-removed triples with all vertices less than or equal to 7664.

Gauging the Independence of a List of Special Vertices As an illustration of a methodology to gauge the independence of a set of vertices, we choose a list of 4 randomly selected special vertices and uniformly at random remove 10% of the triples in the list of triples with vertices less than or equal to some upper bound around say, 7621. This speeds up the SAT solver computations from about a day to about 100 seconds. Each run uses 1 node with 12 cores. Next we record the run time for 10 runs each, of the 16 truth value combinations, for the 4 special vertices. Each of the 10 runs of the SAT solver uses a different reduced list of triples. There are a total of 10 reduced triple lists for the whole process. See Figure 4. For each of the 16 truth value assignments, we compute an empirical mean run time; then, the sample variance of these 16 numbers is then computed. For a set of chosen special vertices, a low variance indicates high independence, since the solver worked approximately equally hard across the different possible truth values.

5.4 Running a Node Pool on Clusters

We have devised a method for simultaneously running a fixed number of nodes, each running an instance of the solver on 12 cores. We are allowed to use up to about 10 of these nodes at a time, after waiting in the queue for some amount of time. See a detailed description of our clusters at the University of South Carolina, Maxwell and Planck, at

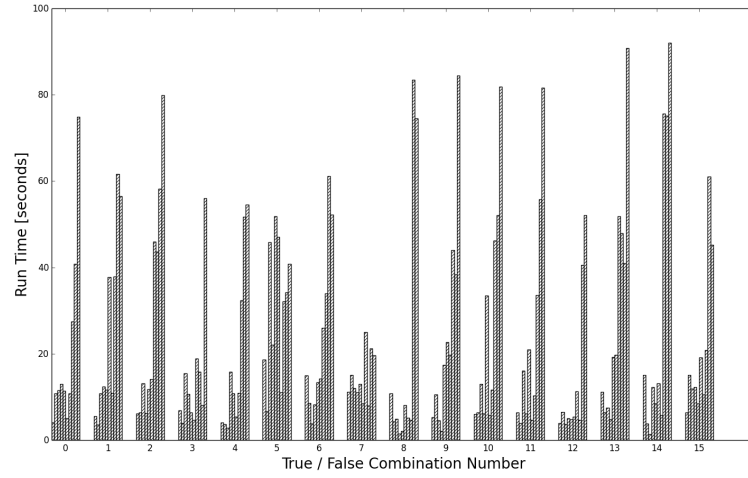


Fig. 4. Run times of 10 runs of each of the 16 possible truth values of the 4 special vertices. The special vertices were chosen uniformly at random.

http://www.sc.edu/about/offices_and_divisions/division_of_information_technology/rci/hpc_resources/index.php

We use the two clusters interchangeably. Their specifications are as follows.

- Maxwell
 - Hardware
 - * 40 GL390 Nodes: 12 cores per node, Intel Xeon 2.4 GHz, 24 GB RAM
 - * 6 SL250: 16 cores per node, Intel Xeon 2.60GHz, 32 GB RAM
 - * 1 DL380 Headnode: 12 core, 48GB RAM
 - * 24TB attached storage
 - Interconnect
 - * QDR Infiniband
 - Software
 - * CentOS
 - * HP CMU cluster management utility
 - * OpenMPI
 - * Torque/Maui scheduler
- Planck
 - Hardware
 - * 20 SL250 Nodes: 12 cores per node, Intel Xeon 2.8 GHz, 24 GB RAM
 - * 15 x 3 NVIDIA M1060 with 240 cores each, 3 x 3
 - * NVIDIA M2070 with 448 cores each
 - * 11 TB attached storage

- Interconnect
 - * QDR Infiniband
- Software
 - * CentOS
 - * HP CMU cluster management utility
 - * OpenMPI
 - * Torque/Maui scheduler

Our parallelization starts from an outer shell script which runs a sequential Python program that creates a fixed number of inner shell scripts, each running 1 instance of Treengeling on one full node. The Python program, which is started by the outer shell script, uses system calls to detect the completions of each of the inner shell scripts by detecting the existence of the output file for each inner shell script. When an inner shell script completes, a new inner shell script is fed to the queue until all the desired instances of Treengeling have been submitted, or until one instance of Treengeling returns a valid coloring.

5.5 Choosing the Number of Special Vertices

We present some evidence that splitting approximately 4 ways using 2 special vertices is fastest, given current technical constraints. See Figure 5, which plots run time against the number of nodes at which the problem was split.

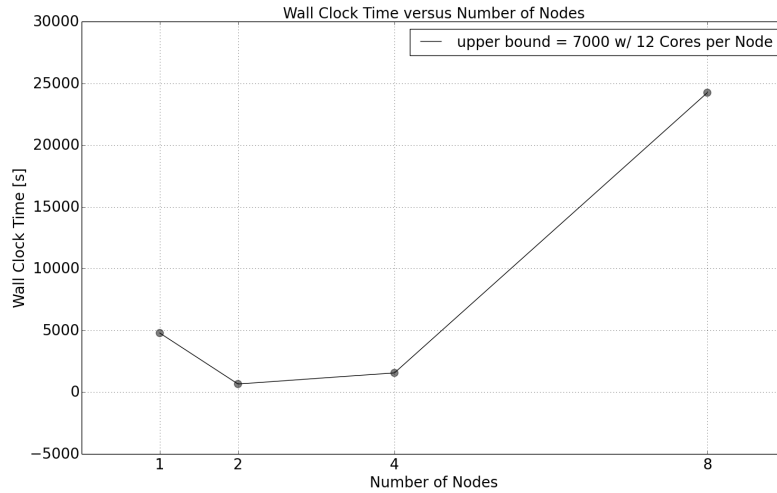


Fig. 5. Some results of scalability testing.

6 Results

6.1 Performance of Treengeling

Treengeling can find solutions for all the Pythagorean triples with integers up to 7620 in about 21 hours. If the list of triples has been pendant removed, Treengeling can find a solution for all triples up to 7650 in about 33 hours.

6.2 Freedom of Coloring and Searching for Patterns

We have looked for some patterns in the large valid colorings found, but have not found much. There seems to be a lot of freedom in how a coloring can be achieved, which seems to be a reason why we have not found any consistent patterns in the colorings.

6.3 A Visualization of the Coloring up to 7664

The coloring in Figure 6 was made by removing the pendants from the list of triples with integers up to 7664, running the solver on the result, and adding back the triples, one at a time, in the reverse order that they were removed and 2-coloring them as they were added back. The final coloring was checked to see that every triple with all integers less than or equal to 7664 is bichromatic.

Integers which do not occur in any triple, with all constituent integers of the triple being less than or equal to 7664, are white. The solver colors every integer which it receives from any triple as either false or true. While some integers might be colorable as either false or true in a particular coloring, the solver will assign some color to each of these integers along with every other integer that it receives.

7 Future Work

7.1 Further Reducing the Input

We are running a sequential Python function on the clusters that systematically checks all possible linear 3-uniform hypergraphs on n vertices to see if any further reductions to PYTH can be made. We have not yet found any “removable” subgraphs that are not pendants, from the triple list up to 7700. We would like to parallelize this sequential operation using Python and MPI, or C and MPI if need be.

Acknowledgements Thanks to Ron Graham, Bill Kay, and Christopher Poirer for helpful discussions and ideas in the development of the present work. Paul Sagona has been helpful with the high performance computing resources at the University of South Carolina.

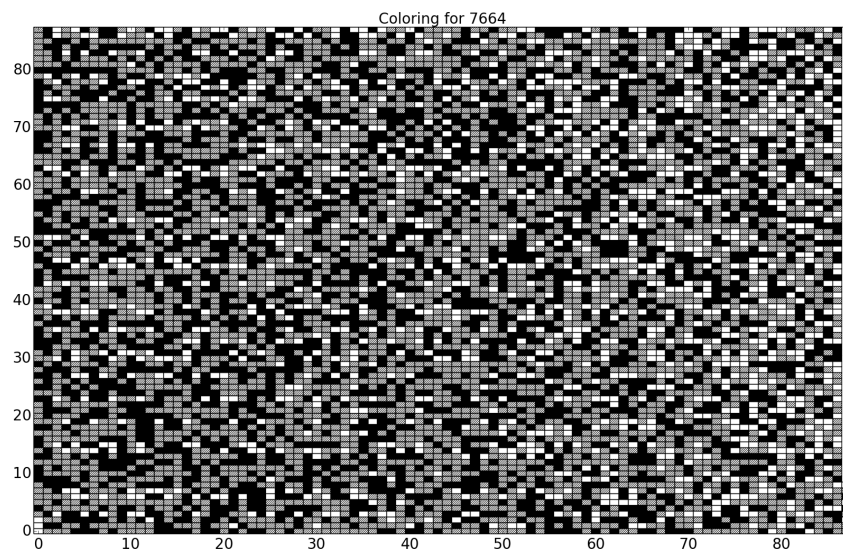


Fig. 6. A coloring of the hypergraph of PYTH induced by the vertex set $[7664]$. False is **grey**. True is **black**. Integers not appearing in the input to the solver are **white**. The integer positions climb vertically by columns from the lower left to the upper right.

References

1. Bruce M. Landman and Aaron Robertson: Ramsey theory on the integers. Student Mathematical Library, **24**. American Mathematical Society, Providence, RI (2004)
2. Joshua Cooper, Michael Filaseta, Joshua Harrington and Daniel White: Colorings of Pythagorean triples within colorings of the positive integers. *Journal of Combinatorics and Number Theory* **6**, pp. 1–16. (2014)
3. Oystein Ore: Theory of graphs. American Mathematical Society Colloquium Publications, Vol. XXXVIII American Mathematical Society, Providence, R.I. (1983)
4. Wacław Sierpiński: Elementary theory of numbers. North-Holland Mathematical Library, 31. North-Holland Publishing Co., Amsterdam; PWN—Polish Scientific Publishers, Warsaw (1988)
5. Charles J. Colbourn and Yuichiro Fujiwara: Small stopping sets in Steiner triple systems, *Crypt. Commun.* **1**, no. 1, pp. 31–46 (2009)
6. Charles J. Colbourn, Marlene J. Colbourn, Alexander Rosa: Completing small partial triple systems. *Discrete Math.* **45**, no. 2-3, pp. 165–179 (1983)
7. Wikipedia contributors: Formulas for generating Pythagorean triples, Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Formulas_for_generating_Pythagorean_triples&oldid=652792835 (accessed April 16, 2015).
8. CNF, <http://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html>
9. Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, Armin Biere: Hardware and Software: Verification and Testing, *Lecture Notes in Computer Science Volume 7261*, 2012, pp 50-65 Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads.